

IBM Research

The IBM High Performance Computing Toolkit on BlueGene/L

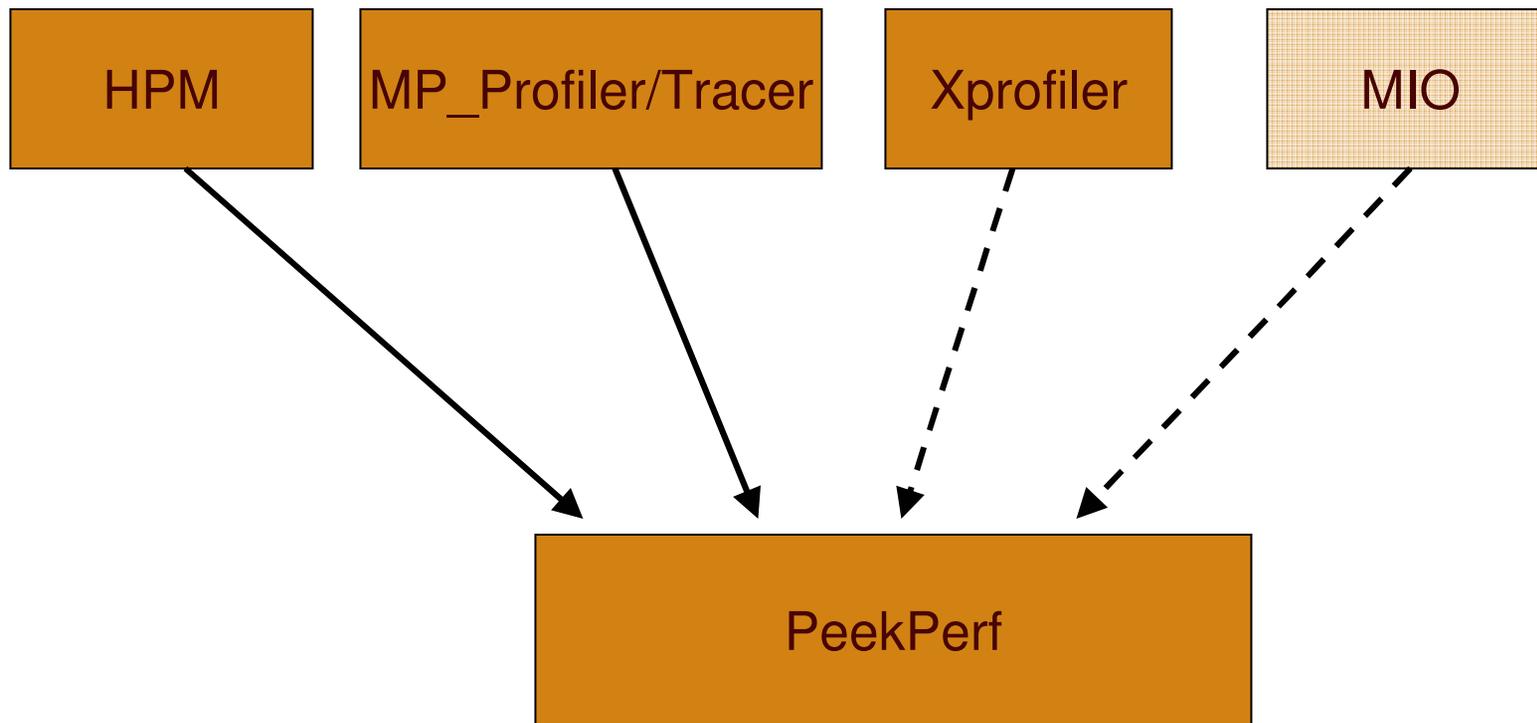
I-Hsin Chung
Guojing Cong

ihchung@us.ibm.com
gcong@us.ibm.com

IBM High Performance Computing Toolkit on BG/L

- **MPI performance: MP_Profiler**
- **CPU performance: Xprofiler, HPM**
- **Visualization and analysis: PeekPerf**
- **Modular I/O: MIO**

IBM HPCT Overview



Message-Passing Performance:

■ **MP_Profiler Library**

- Captures “summary” data for MPI calls
- Source code traceback
- User **MUST** call MPI_Finalize() in order to get output files.
- No changes to source code
 - MUST compile with `-g` to obtain source line number information

■ **MP_Tracer Library**

- Captures “timestamped” data for MPI calls
- Source traceback

- **Available at /bgl/home1/ihchung/hpct_bgl/mp_profiler on *BG/L front end node***

Compiling and Linking Example

BGL=/bgl/BlueLight/ppcfloor

CC=\$(BGL)/blrts-gnu/powerpc-bgl-blrts-gnu/bin/gcc

CFLAG= -I \$(BGL)/bglsys/include

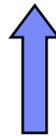
MPI_LIB= -L \$(BGL)/bglsys/lib -lmpich.rts -lmsglayer.rts -lrts.rts -ldevices.rts

TRACE_LIB= -L \$(MP_PROFILER) -lmpitrace.rts

BINUTILS_LIB= -L \$(BINUTILS) -lbfd -liberty

target: source.c

\$(CC) -o \$@ \$< \$(CFLAG) **\$(TRACE_LIB)** **\$(MPI_LIB)** \$(BINUTIL_LIB)



\$(TRACE_LIB) has to precede \$(MPI_LIB)

MP_Profiler Output with Peekperf

The screenshot displays the IBM ACTC PeekPerf Main Window. The main window shows a list of MPI application functions with their call counts and wall clock times. A secondary window, 'Metric Browser: MPI_Send_130', provides a detailed view of the selected function, including a table of task metrics and the corresponding Fortran code snippet.

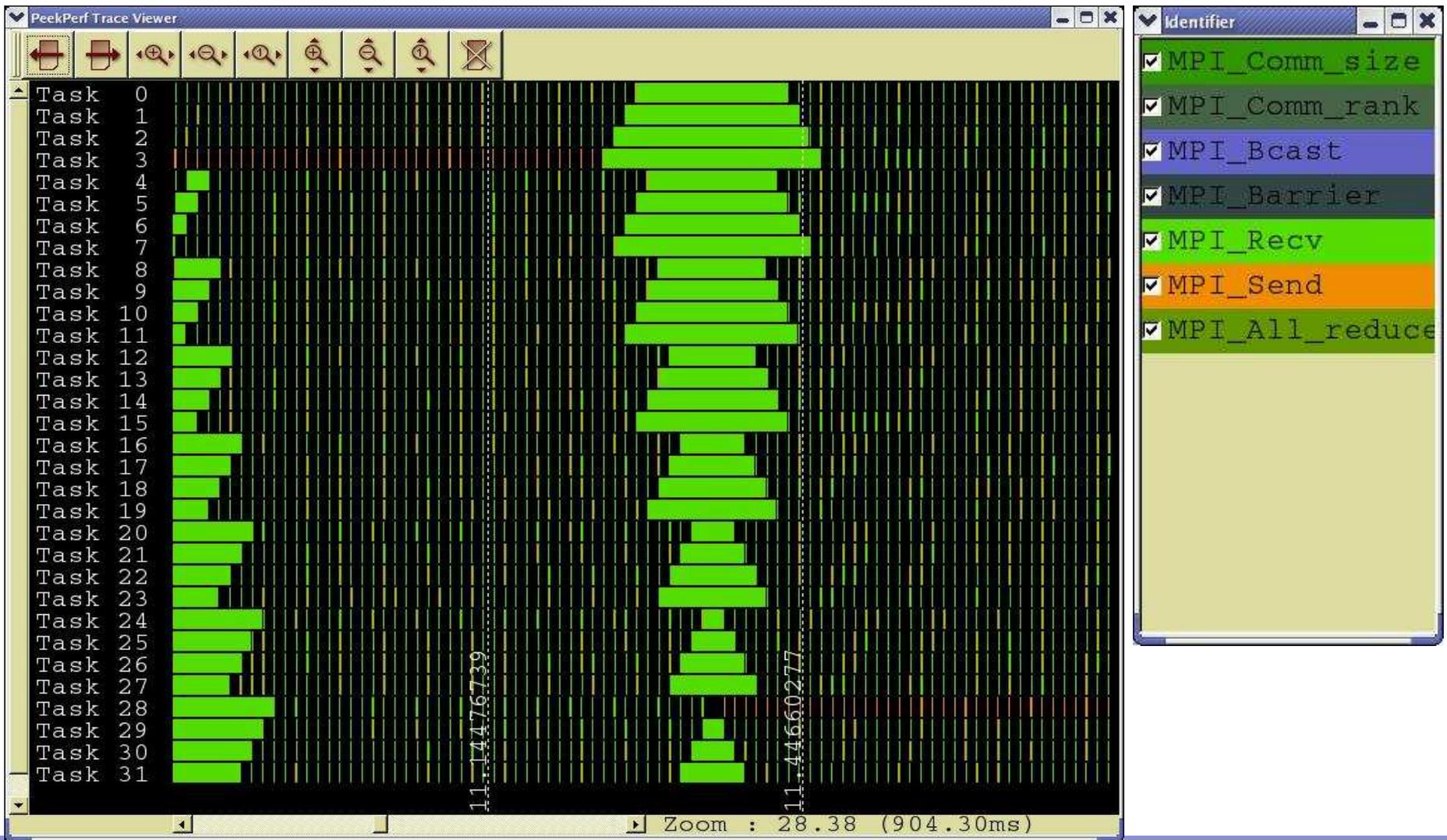
Task	Message Size	Count	WallClock [Max]	Transferred Bytes	Call Count [Max]	WallClock
0	(8) 16K ... 64K	1920	0.513883	2.21184e+07	1920	0.513883
1	(8) 16K ... 64K	2880	0.549085	3.31776e+07	2880	0.549085
2	(8) 16K ... 64K	2880	0.551206	3.31776e+07	2880	0.551206
3	(8) 16K ... 64K	1920	0.516694	2.21184e+07	1920	0.516694
4	(8) 16K ... 64K	2880	0.632482	3.31776e+07	2880	0.632482
5	(8) 16K ... 64K	3840	0.654542	4.42368e+07	3840	0.654542
6	(8) 16K ... 64K	3840	0.651453	4.42368e+07	3840	0.651453
7	(8) 16K ... 64K	2880	0.625413	3.31776e+07	2880	0.625413
8	(8) 16K ... 64K	2880	0.593683	3.31776e+07	2880	0.593683
9	(8) 16K ... 64K	3840	0.643496	4.42368e+07	3840	0.643496
10	(8) 16K ... 64K	3840	0.640881	4.42368e+07	3840	0.640881
11	(8) 16K ... 64K	2880	0.589392	3.31776e+07	2880	0.589392
12	(8) 16K ... 64K	2880	0.553126	3.31776e+07	2880	0.553126

```

return
end

subroutine rcv_real(orig, value, size, ta
implicit none
    
```

MP_Profiler - Traces



MP_Profiler Message Size Distribution

MPI Function	#Calls	Message Size	#Bytes	Walltime	MPI Function	#Calls	Message Size	#Bytes	Walltime
MPI_Comm_size	1 (1)	0 ... 4	0	1E-07	MPI_Irecv	2 (1)	0 ... 4	3	4.7E-06
MPI_Comm_rank	1 (1)	0 ... 4	0	1E-07	MPI_Irecv	2 (2)	5 ... 16	12	1.4E-06
MPI_Isend	2 (1)	0 ... 4	3	0.000006	MPI_Irecv	2 (3)	17 ... 64	48	1.5E-06
MPI_Isend	2 (2)	5 ... 16	12	1.4E-06	MPI_Irecv	2 (4)	65 ... 256	192	2.4E-06
MPI_Isend	2 (3)	17 ... 64	48	1.3E-06	MPI_Irecv	2 (5)	257 ... 1K	768	2.6E-06
MPI_Isend	2 (4)	65 ... 256	192	1.3E-06	MPI_Irecv	2 (6)	1K ... 4K	3072	3.4E-06
MPI_Isend	2 (5)	257 ... 1K	768	1.3E-06	MPI_Irecv	2 (7)	4K ... 16K	12288	7.1E-06
MPI_Isend	2 (6)	1K ... 4K	3072	1.3E-06	MPI_Irecv	2 (8)	16K ... 64K	49152	2.23E-05
MPI_Isend	2 (7)	4K ... 16K	12288	1.3E-06	MPI_Irecv	2 (9)	64K ... 256K	196608	9.98E-05
MPI_Isend	2 (8)	16K ... 64K	49152	1.3E-06	MPI_Irecv	2 (A)	256K ... 1M	786432	0.00039
MPI_Isend	2 (9)	64K ... 256K	196608	1.7E-06	MPI_Irecv	1 (B)	1M ... 4M	1048576	0.000517
MPI_Isend	2 (A)	256K ... 1M	786432	1.7E-06	MPI_Waitall	21 (1)	0 ... 4	0	1.98E-05
MPI_Isend	1 (B)	1M ... 4M	1048576	9E-07	MPI_Barrier	5 (1)	0 ... 4	0	7.8E-06

Environment Flags

- **TRACELEVEL**
 - Level of trace back the caller in the stack
 - Used to skipped wrappers
 - Default: 0
- **TRACE_TEXTONLY**
 - If set to “1”, plain text output is generated
 - Otherwise, a viz file is generated
- **TRACE_PERFILE**
 - If set to “1”, the output is shown for each source file
 - Otherwise, output is a summary of all source files
- **TRACE_PERSIZE**
 - If set to “1”, the static for a function is shown for every message size
 - Otherwise, summary for all message sizes is given

Xprofiler

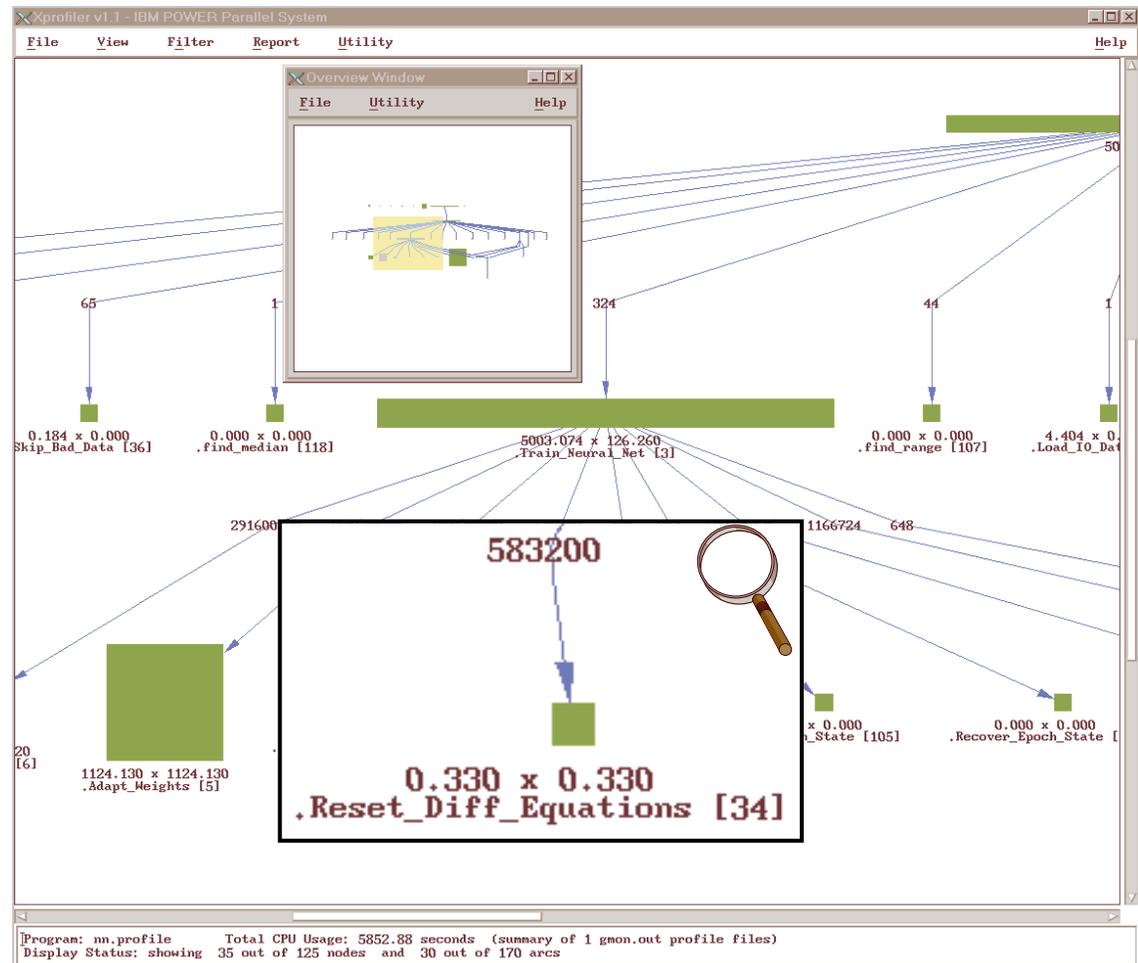
- **CPU profiling tool similar to gprof**
- **Can be used to profile both serial and parallel applications**
- **Use procedure-profiling information to construct a graphical display of the functions within an application**
- **Provide quick access to the profiled data and helps users identify functions that are the most CPU-intensive**
- **Based on sampling (support from both compiler and kernel)**
- **Charge execution time to source lines and show disassembly code**
- ***[/bgl/home1/ihchung/hpct_bgl/xprofiler](#) on [BG/L front end node](#)***

Running Xprofiler

- **Compile the program with -pg**
- **Run the program**
- **gmon.out file is generated (MPI applications generate gmon.out.1, ..., gmon.out.n)**
- **Run Xprofiler**

Xprofiler: Main Display

- **Width of a bar:** time including called routines
- **Height of a bar:** time excluding called routines
- **Call arrows** labeled with number of calls
- **Overview window** for easy navigation (View → Overview)



Xprofiler: Source Code Window

- **Source code window displays source code with time profile (in ticks=.01 sec)**
- **Access**
 - Select function in main display
 - → context menu
 - Select function in flat profile
 - → Code Display
 - → Show Source Code

line	no. ticks per line	source code
202		/*-----*/
203		/* use 2x-unrolling of the outer two loops */
204		/*-----*/
205	4	for (i=i0; i<i0+is-1; i+=2)
206		{
207	8	for (j=j0; j<j0+js-1; j+=2)
208		{
209	1	t11 = c[i*n+j];
210	5	t12 = c[i*n+j+1];
211	5	t21 = c[(i+1)*n+j];
212	19	t22 = c[(i+1)*n+(j+1)];
213		for (k=k0; k<k0+ks; k++)
217	229	t21 = t21 + a[(i+1)*n+k]*bt[j*n+k];
218	144	t22 = t22 + a[(i+1)*n+k]*bt[(j+1)*n+k];
219		}
220	7	c[i*n+j] = t11;
221		c[i*n+j+1] = t12;
222	3	c[(i+1)*n+j] = t21;
223	5	c[(i+1)*n+(j+1)] = t22;
224		}
225		for (j=j; j<j0+js; j++)
226		{
227		t11 = c[i*n+j];
228		t21 = c[(i+1)*n+j];
229		for (k=k0; k<k0+ks; k++)
230		{
231		t11 = t11 + a[i*n+k]*bt[j*n+k];
232		t21 = t21 + a[(i+1)*n+k]*bt[j*n+k];
233		}
234		c[i*n+j] = t11;
235		c[(i+1)*n+j] = t21;
236		}
237		}

Search Engine: (regular expressions supported)

t21

Xprofiler - Disassembler Code

Disassembler Code for .calc3 [3]

address	no. ticks per instr.	instruction	assembler code	source code
10002E18	81	FCC4287C	fnms 6,4,1,5	
10002E1C	64	CCF70008	lfdu 7,0x8(23)	POLD(I, J) = P(I, J)+ALPHA*(PNEW(I, J)-
10002E20	187	C90C0008	lfd 8,0x8(12)	
10002E24	53	C9750008	lfd 11,0x8(21)	UOLD(I, J) = U(I, J)+ALPHA*(UNEW(I, J)-
10002E28	89	FD63582A	fa 11,3,11	
10002E2C	63	FD28387C	fnms 9,8,1,7	POLD(I, J) = P(I, J)+ALPHA*(PNEW(I, J)-
10002E30	4	DD5B0008	stfdu 10,0x8(27)	U(I, J) = UNEW(I, J)
10002E34		C9540008	lfd 10,0x8(20)	VOLD(I, J) = V(I, J)+ALPHA*(VNEW(I, J)-
10002E38	113	FCCA302A	fa 6,10,6	
10002E3C	27	C8760008	lfd 3,0x8(22)	POLD(I, J) = P(I, J)+ALPHA*(PNEW(I, J)-
10002E40	87	FD8012FA	fma 12,0,11,2	UOLD(I, J) = U(I, J)+ALPHA*(UNEW(I, J)-
10002E44	35	DCB90008	stfdu 5,0x8(25)	V(I, J) = VNEW(I, J)
10002E48	4	FC63482A	fa 3,3,9	POLD(I, J) = P(I, J)+ALPHA*(PNEW(I, J)-
10002E4C	12	CD5A0008	lfdu 10,0x8(26)	UOLD(I, J) = U(I, J)+ALPHA*(UNEW(I, J)-
10002E50	62	FCC021BA	fma 6,0,6,4	VOLD(I, J) = V(I, J)+ALPHA*(VNEW(I, J)-
10002E54	36	C85B0008	lfd 2,0x8(27)	UOLD(I, J) = U(I, J)+ALPHA*(UNEW(I, J)-
10002E58	244	DCEC0008	stfdu 7,0x8(12)	P(I, J) = PNEW(I, J)
10002E5C	28	FD0040FA	fma 8,0,3,8	POLD(I, J) = P(I, J)+ALPHA*(PNEW(I, J)-
10002E60		C8990008	lfd 4,0x8(25)	VOLD(I, J) = V(I, J)+ALPHA*(VNEW(I, J)-
10002E64	316	DCD40008	stfdu 6,0x8(20)	
10002E68	29	FC62507C	fnms 3,2,1,10	UOLD(I, J) = U(I, J)+ALPHA*(UNEW(I, J)-

Search Engine: (regular expressions supported)

LIBHPM

- **Instrumentation library**
- **Provides performance information for instrumented program sections**
- **Supports multiple (nested) instrumentation sections**
- **Multiple sections may have the same ID**
- **Run-time performance information collection**
- **Based on bgl_perfctr layer – can be eliminated in BG/P**
- **Available at /bgl/home1/ihchung/hpct_bgl/hpm on *BG/L front end node***

Event Sets

- **16 sets (0-15); 328 events**
- **Information for**
 - Time
 - FPU (0,1)
 - L3 memory
 - Processing Unit (0,1)
 - Tree network
 - Torus network
- **For detailed names and descriptions: [event_sets.txt](#)**

Functions

- **hpmInit(taskID, progName) / f_hpminit(taskID, progName)**
 - taskID is an integer value indicating the node ID.
 - progName is a string with the program name.
- **hpmStart(instID, label) / f_hpmstart(instID, label)**
 - instID is the instrumented section ID. It should be > 0 and ≤ 100 (can be overridden)
 - Label is a string containing a label, which is displayed by PeekPerf.
- **hpmStop(instID) / f_hpmstop(instID)**
 - For each call to hpmStart, there should be a corresponding call to hpmStop with matching instID
- **hpmTerminate(taskID) / f_hpmterminate(taskID)**
 - This function will generate the output. If the program exits without calling hpmTerminate, no performance information will be generated.

Functions (continued)

- **hpmGetTimeAndCounters(numCounters, time, values)**
/ f_GetTimeAndCounters (numCounters, time, values)
 - returns the time in seconds and counts since the call to hpmInit.
 - numCounters: integer indicating the number of counters to be accessed.
 - time: double precision float
 - values: “long long” vector of size “numCounters”.
- **hpmGetCounters(values) / f_hpmGetCounters (values)**
 - Similar to hpmGetTimeAndCounters
 - only returns the total counts since the call to hpmInit

Example of Use

- **C / C++**

declaration:

```
#include "libhpm.h"
```

use:

```
hpmInit( taskID, "my program" );  
hpmStart( 1, "outer call" );  
do_work();  
hpmStart( 2, "computing meaning of life" );  
do_more_work();  
hpmStop( 2 );  
hpmStop( 1 );  
hpmTerminate( taskID );
```

- **Flags**

- Compiling: -I\$(HPM_DIR)/include
- Linking: -L\$(BGL_FLOOR)/bglsys/lib -L\$(HPM_DIR)/lib -lhpm.rts -lm -lbgl_perfctr.rts

Example of Use (continued)

- Fortran

declaration:

```
#include "f_hpm.h"
```

use:

```
call f_hpminit( taskID, "my program" )
```

```
call f_hpmstart( 1, "Do Loop" )
```

```
do ...
```

```
  call do_work()
```

```
  call f_hpmstart( 5, "computing meaning of life" );
```

```
  call do_more_work();
```

```
  call f_hpmstop( 5 );
```

```
end do
```

```
call f_hpmstop( 1 )
```

```
call f_hpmterminate( taskID )
```

Output

- **Summary report for each task**

- perfhpm<taskID>.<pid>

libhpm (V 2.6.0) summary

Total execution time of instrumented code (wall time): 0.143824 seconds

Instrumented section: 3 - Label: job 1 - process: 1

file: sanity.c, lines: 33 <--> 70

Count: 1

Wall Clock Time: 0.143545 seconds

BGL_FPU_ARITH_MULT_DIV (Multiplication and divisions, fmul, fmul, fdiv, fdivs (Book E mul, div)) : 0

BGL_FPU_LDST_DBL_ST (...) : 23

...

BGL_UPC_L3_WRBUF_LINE_ALLOC (Write buffer line was allocated) :1702

...

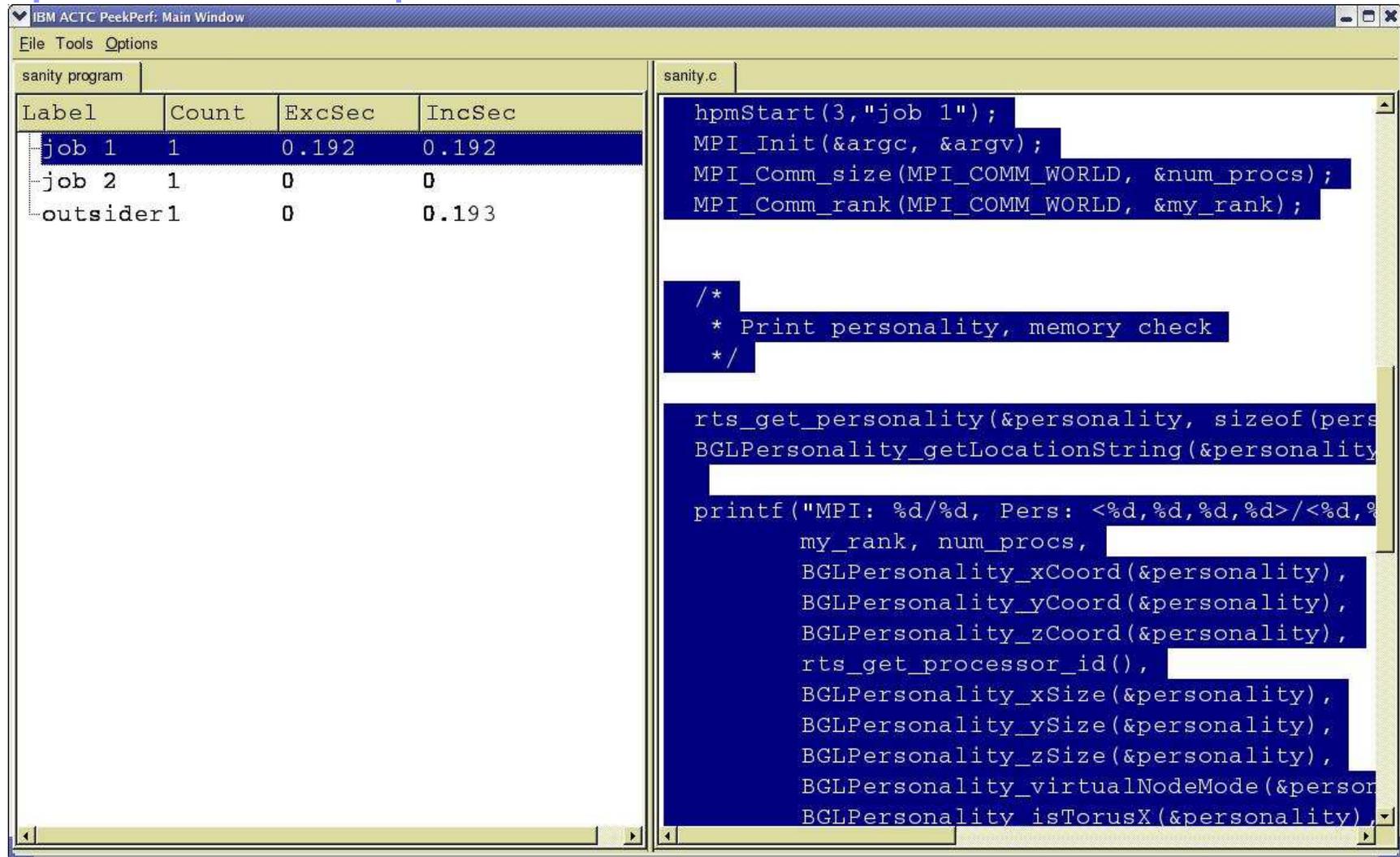
- **Peekperf performance file**

- hpm<taskID>_<progName>_<pid>.viz

- **Table performance file**

- tb_hpm<taskID>.<pid>

Output with Peekperf



The screenshot shows the IBM ACTC PeekPerf Main Window. The window is divided into two panes. The left pane, titled 'sanity program', displays a performance table with the following data:

Label	Count	ExcSec	IncSec
job 1	1	0.192	0.192
job 2	1	0	0
outsider1	0	0	0.193

The right pane, titled 'sanity.c', displays the source code for the program. The code includes MPI initialization and personality-related functions. The code is as follows:

```
hpmStart(3,"job 1");
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

/*
 * Print personality, memory check
 */

rts_get_personality(&personality, sizeof(personality));
BGLPersonality_getLocationString(&personality);
printf("MPI: %d/%d, Pers: <%d,%d,%d,%d>/<%d,%d,%d,%d>\n",
my_rank, num_procs,
BGLPersonality_xCoord(&personality),
BGLPersonality_yCoord(&personality),
BGLPersonality_zCoord(&personality),
rts_get_processor_id(),
BGLPersonality_xSize(&personality),
BGLPersonality_ySize(&personality),
BGLPersonality_zSize(&personality),
BGLPersonality_virtualNodeMode(&personality),
BGLPersonality_isTorusX(&personality));
```

Environment Flags

- **HPM_EVENT_SET**
 - Select the event set to be recorded
 - Integer (0 – 15)
- **HPM_NUM_INST_PTS**
 - Overwrite the default of 100 instrumentation sections in the app.
 - Integer value > 0
- **HPM_WITH_MEASUREMENT_ERROR**
 - Deactivate the procedure that removes measurement errors.
 - True or False (0 or 1).
- **HPM_OUTPUT_NAME**
 - Define an output file name different from the default.
 - String
- **HPM_VIZ_OUTPUT**
 - Indicate if “.viz” file (for input to PeekPerf) should be generated or not.
 - True or False (0 or 1).
- **HPM_TABLE_OUTPUT**
 - Indicate table text file should be generated or not.
 - True or False (0 or 1).

Related Work

- **PARAVER (UPC)**
 - Performance Visualization and Analysis Tool
 - Flexibility to represent traces from different environment
 - MPI trace, HW performance counter info (based on PAPI)
- **PAPI (Univ. of Tennessee)**
 - **P**erformance **A**pplication **P**rogramming **I**nterface
 - Design, standardize, and implement a portable and efficient API to access the hardware performance counters
- **Kojak (Juelich)**
 - Generic automatic performance analysis environment for parallel programs

Related Work (continued)

- **TAU (Univ. of Oregon)**
 - **Tuning and Analysis Utilities**
 - Program and performance analysis tool framework for high-performance parallel and distributed computing
- **mpiP (LLNL/ORNL)**
 - Lightweight profiling library for MPI applications
 - Only collects statistical information about MPI functions
 - With less overhead and have much less data than tracing tools
 - Only uses communication during report generation stage

Future work

- **Mp_profiler**
 - Scalability
- **Xprofiler**
 - Improved GUI
 - Integration with other platforms
- **HPM**
 - Useful derived metrics and verifications
 - Hpmcount / Hpmstat ?
 - Integration with other platforms